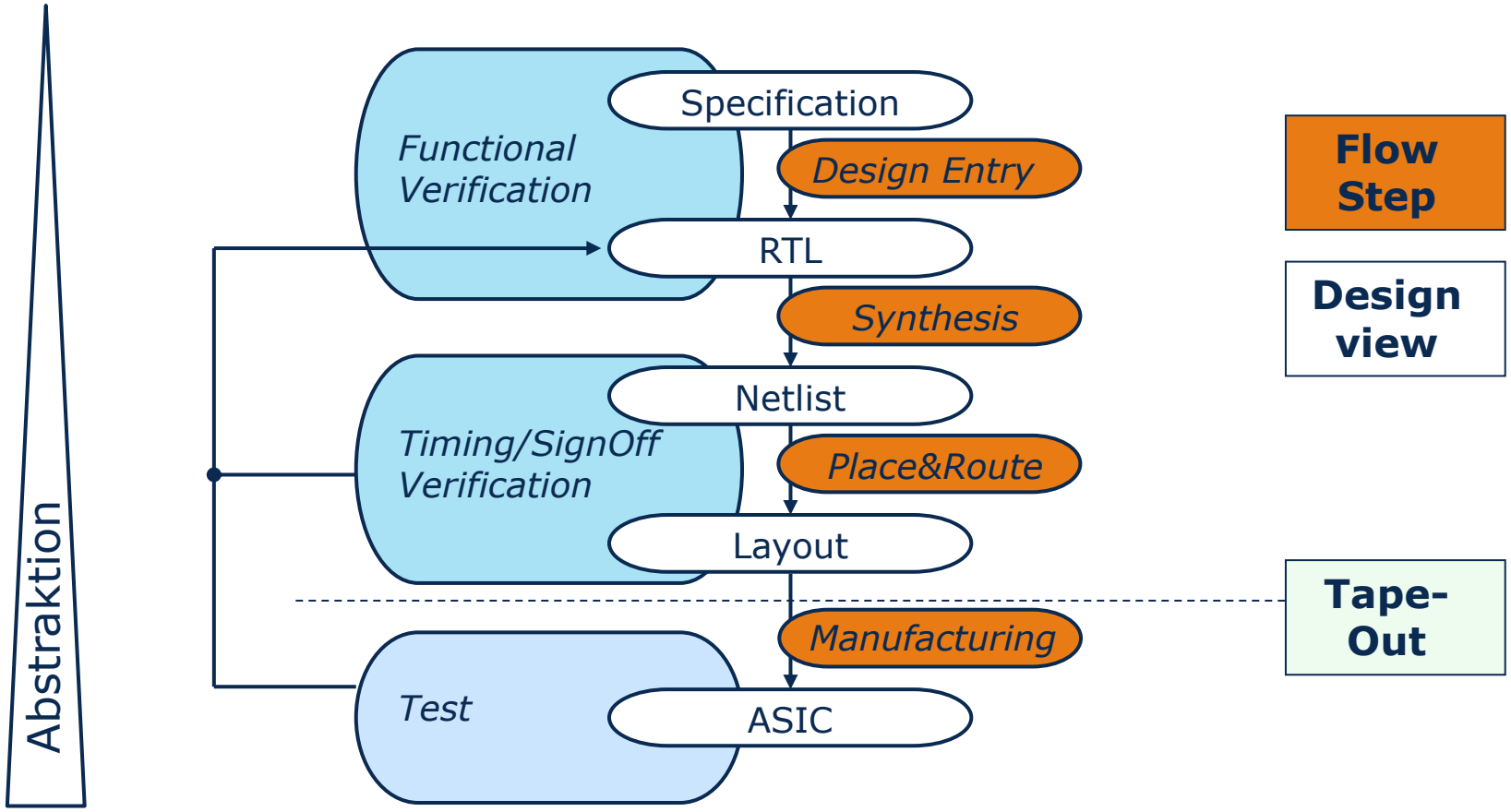
Verifikation

Design Flow: Semi-Custom Digital Design



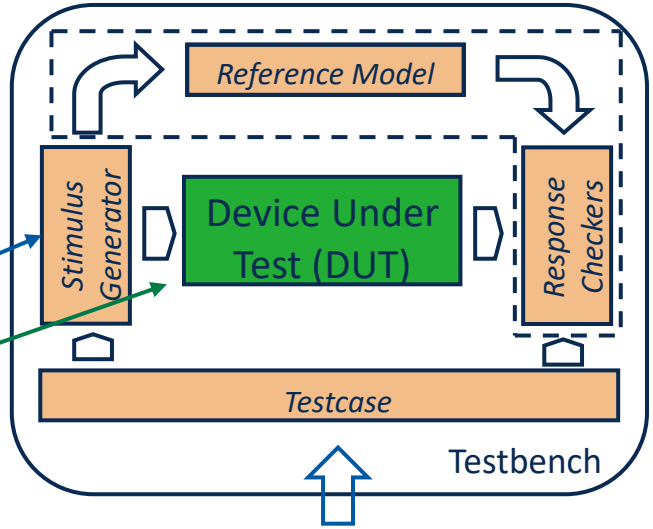
- Nachweis der Übereinstimmung von Schaltungsfunktionalität und Spezifikation
- ggf. Vergleich der Übereinstimmung mit einem Referenzmodell

- Verifikation findet in verschiedenen Hierarchieebenen statt
  - Block → Modul → System

- Verifikationsmethoden
  - **Verifikation durch Schaltungssimulation**
  - Formale Verifikation
  - Hardware Emulation/Prototyping (z.B. mit FPGAs)

- Bei aktuellen System-on-Chip Designs werden **bis zu 80%** der Entwurfszeit- und Ressourcen zur Verifikation verwendet

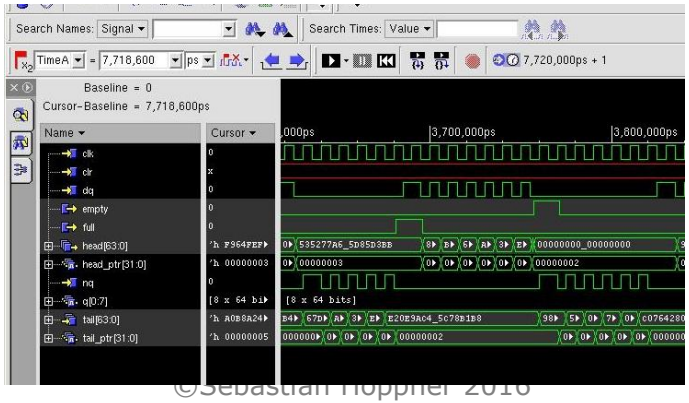
```
module testbench ()

   //Signals
   reg …
   wire …

   `include "my_tasks.v"
   `include "my_functions.v"

   `include „stim_clk_gen.v"

   my_module dut (
      .clk_i(clk),
      .reset_q_i(reset_q),
      .a_i(a),
      .b_o(b));

  `include „testcase.v"

end
endmodule
```




```
// testcase.v
// testcase.v
// testcase.v

initial begin
…
//testcase code
…
end
```

individuelle Simulationsverzeichnisse

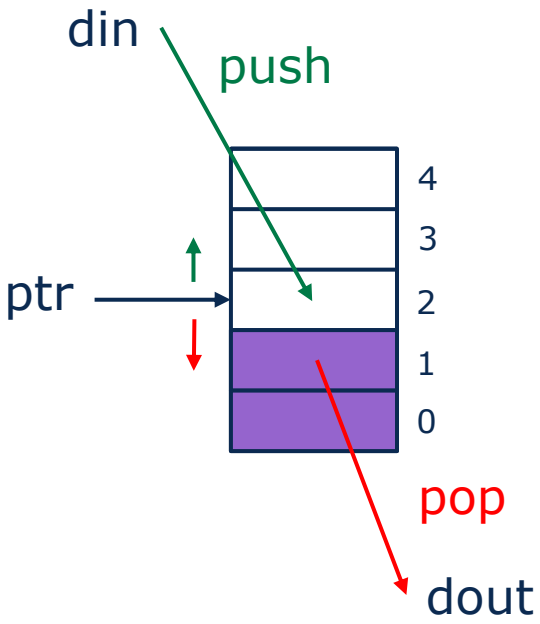
- Nutzung von Waveforms zur Verifikation Schaltungsfunktionalität
  - Vorteile ☺
    - Intuitiver Ansatz
    - Visualisiert das Verhalten der Schaltung
    - Vermittelt Verständnis zur Funktion des Schaltung
    - Erleichtert Debugging
  - Nachteile ☹
    - Sehr zeitaufwändig bei komplexen Systemen
    - **Nicht automatisierbar** und **reproduzierbar**
    - Grad der Verifikationsabdeckung nicht messbar



- In der Beschreibung von Schaltungsblöcken können **Bedingungen** für das erwartete Schaltungsverhalten formuliert werden
  - Einbringen von detaillierten Spezifikationen auf RTL-Level möglich
  - Verifikation wird bereits bei der RTL Implementierung unterstützt
- Prüfung der Bedingungen findet nur im Schaltungsmodell statt, nicht in der realisierten Hardware
- Nutzung von **nicht synthesefähigen** HDL Konstrukten
- Überprüfung dieser Bedingungen bei der **Simulation**
- Ausgaben bei Verletzung der Bedingungen

Beispiel: Stack mit Assertions

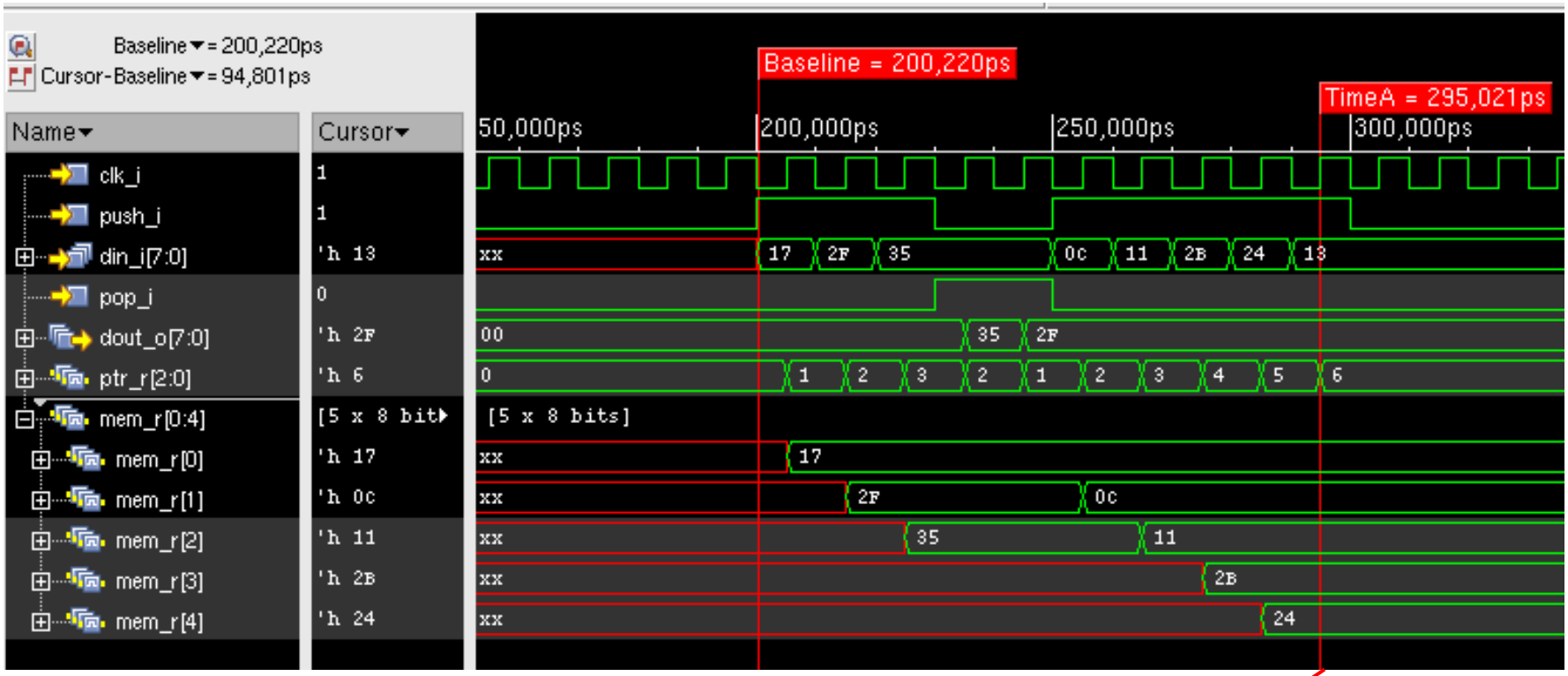
```
module stack
(reset_q_i,clk_i,push_i,pop_i,din_i,dout_o);

parameter DBITS=8;
parameter PTRBITS=3;
parameter DEPTH=5;
input  reset_q_i, clk_i, push_i, pop_i;
input  [DBITS-1:0]        din_i;
output [DBITS-1:0]        dout_o;

reg [DBITS-1:0] mem_r[0:DEPTH-1];
reg [PTRBITS-1:0] ptr_r;
reg [DBITS-1:0] do_r;

wire [PTRBITS-1:0] ptr_m1, ptr_p1;
assign ptr_m1=ptr_r-1;
assign ptr_p1=ptr_r+1;

always @(posedge clk_i) begin
   if (push_i==1'b1) mem_r[ptr_r]<=din_i;
end
…
```



Beispiel: Stack mit Assertions

```
…
always @(posedge clk_i or negedge reset_q_i) begin
   if (reset_q_i==1'b0) begin
      ptr_r<=0;
      do_r <=0;
   end
   else begin
      if(push_i==1'b1) begin
         // synopsys translate_off
         if(ptr_r==DEPTH) $display("WARNING: STACK OVERFLOW at time %e",$realtime);
         // synopsys translate_on
         ptr_r<=ptr_p1;
      end
   else if (pop_i==1'b1) begin
      // synopsys translate_off
      if(ptr_r==0) $display("WARNING: STACK UNDERFLOW at time %e",$realtime);
      // synopsys translate_on
      ptr_r<=ptr_m1;
      do_r<=mem_r[ptr_m1];
   end
   // synopsys translate_off
   if(push_i&&pop_i) $display("WARNING: SIMULTANEOUS PUSH & POP at time %e",$realtime);
   // synopsys translate_on
   end
end
assign dout_o=do_r;
endmodule
```

Beispiel: Stack mit Assertions




```
ncsim> input -quiet .reinvoke.sim
ncsim> file delete .reinvoke.sim
ncsim> run
DONE
WARNING: STACK OVERFLOW at time 2.950000e+02
DONE
Simulation complete via $finish(1) at time 1 US + 0
```

„Self-checking“ Testbenches

- Check: Überprüfen einer Schaltungsausgabe
- Vergleich mit
  - Erwartungswert
  - Spezifikation
  - Referenzmodell
- Formulierung in der Testbench oder im Testcase
- Zählen der Checks und der fehlgeschlagenen Checks (errors)
- Generierung eines finalen Reports

- Vorteile:
  - Schnelle Wiederholbarkeit der Verifikation bei Entwurfsänderungen

Beispiel: Checks mit Referenzmodell

```
//testbench
…
reg [3:0] add_a,add_b;
reg add_cin;
wire [3:0] add_sum, add_sum_ref;
wire add_cout, add_cout_ref;
wire sum_ref;
integer checkcount=0;
integer errorcount=0;

adder adder_i0 (
   .sum_o(add_sum),
   .c_o(add_cout),
   .c_i(add_cin),
   .a_i(add_a),
   .b_i(add_b)) ;

//Reference Model
assign {add_cout_ref,add_sum_ref}=add_a+add_b+add_cin;

`include "testcase.v"
```

```
task check_add;
input [3:0] a;
input [3:0] b;
input cin;
begin
   checkcount=checkcount+1;
   add_a=a;
   add_b=b;
   add_cin=cin;
   #(CLKPERIODE)
   if ((add_sum==add_sum_ref)&&
      (add_cout==add_cout_ref))
   begin
      $display("check: %d %d %d  PASS",a,b,cin);
   end
   else begin
      $display("check: %d %d %d  FAIL",a,b,cin);
      errorcount=errorcount+1;
   end
end
endtask
```

```
//testcase.v
initial begin
  #200
  check_add(3,5,0);
  check_add(6,5,0);
…
end
```

```
check: 3 5 0 PASS
check: 6 5 0 PASS
…
```

Beispiel: Referenzmodell als Tabelle

- Implementierung des „Referenzmodels“ als Tabelle

```
//Beispiel Zustandsübergangstabelle

…
//Instanz der Zustandübergangslogik
…

reg [1:0] state_vec[0:15]
reg [1:0] input_vec[0:15]
reg [1:0] next_state_vec[0:15]
integer i;

initial begin
    state_vec =      '{ 2'b00, 2'b00, 2'b00, 2'b00, 2'b01, 2'b01, … };
    input_vec =      ,{ 2'b00, 2'b01, 2'b10, 2'b11, 2'b00, 2'b01, … };
    next_state_vec = '{ 2'b00, 2'b00, 2'b10, 2'b10, 2'b11, 2'b00, … };
end

for (i=0;i<16;i=i+1) begin
          state=state_vec[i];
          in=input_vec[i]
          if (state_nxt==next_state_vec[i]) $display("check: PASS");
          else                              $display("check: FAIL");
end
…
endmodule
```

inputs

expected outputs

Verifikation von sequentiellen Abläufen

- Bereitstellen der Eingangsdaten
  - Memory Laden
  - Setzen von Inputs
- Starten des Schaltungsblocks
- Warten bis fertig
- Abholen der Ausgangsdaten
  - Speicher lesen
- Ergebnisse Vergleichen
  - Vergleich mit Referenzdaten

```
//testcase.v
integer i;
initial begin
   for (i=0;i<CHECKS;i=i+1) begin
      load_mem_data(i);
      set_inputs(i);
      run_dut;
      get_mem_data;
      compare_result;
   end
end
```

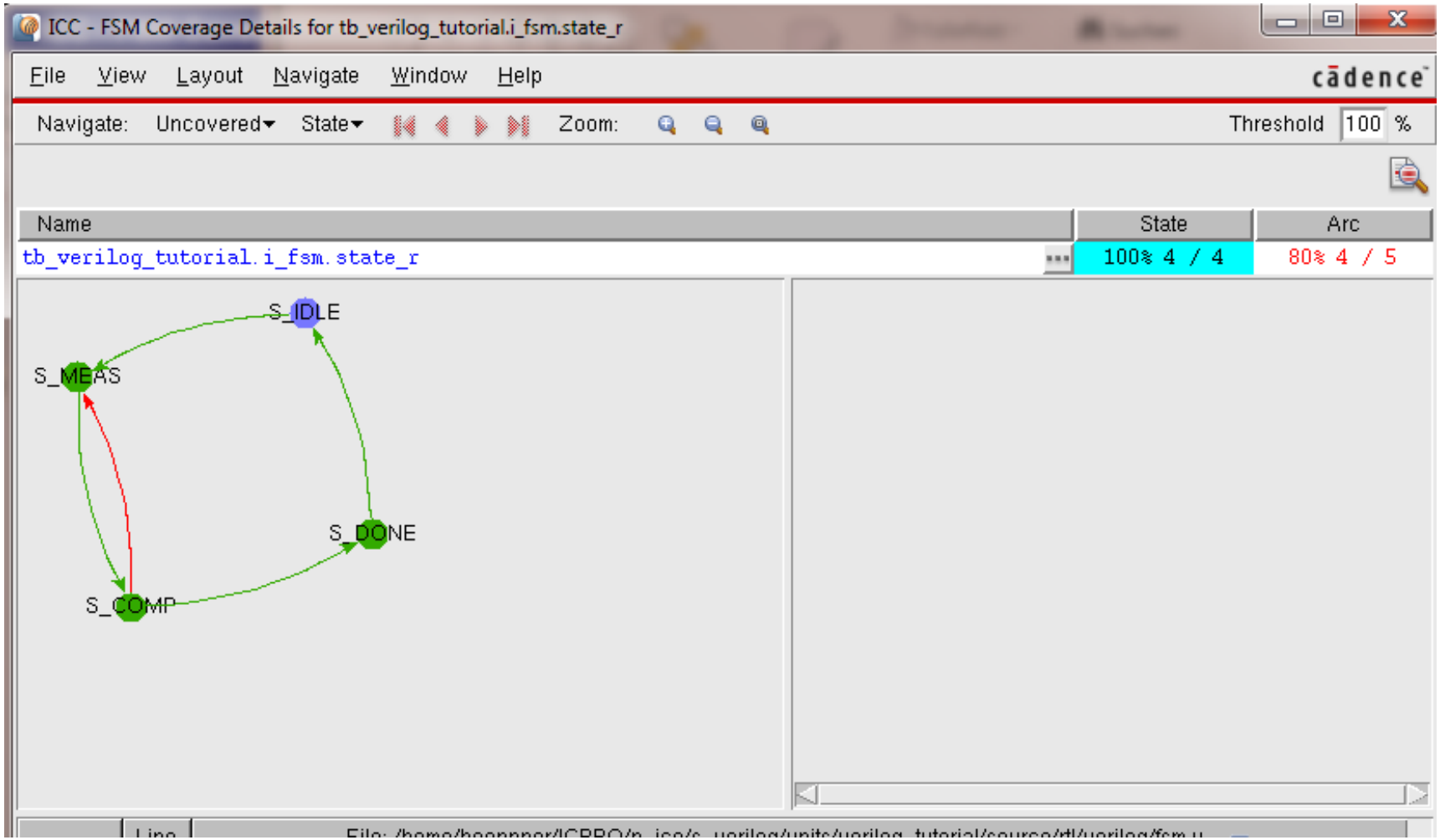
Testbench Note

```
task printTestcaseResults; begin
$write("\n");
if ( checkcount == 0 )         begin
   $display(" #   #  #   #  #  #  #   #   ###   #   #  #   #");
   $display(" #   #  ##  #  # #   ##  #  #   #  #   #  ##  #");
   $display(" #   #  # # #  ###   # # #  #   #  # # #  # # #");
   $display(" #   #  #  ##  # #   #  ##  #   #  # # #  #  ##");
   $display("  ###   #   #  #  #  #   #   ###   ## ##  #   #");
   $display("\nTUD_TESTBENCH:WARNING:SIMUNKNOWN: Test result Unknown");
end
else if ( errorcount > 0 )         begin
   $display(" #####   ###   #  #      #####  #### ");
   $display(" #      #   #  #  #      #      #   #");
   $display(" ###    #####  #  #      ###    #   #");
   $display(" #      #   #  #  #      #      #   #");
   $display(" #      #   #  #  #####  #####  #### ");
   $display("\nTUD_TESTBENCH:ERROR:SIMFAIL: Test FAILed");
end
else        begin
   $display(" ####    ###    ####  ####  #####  #### ");
   $display(" #   #  #   #  #     #     #      #   #");
   $display(" ####   #####   ###   ###  ###    #   #");
   $display(" #      #   #      #     #  #      #   #");
   $display(" #      #   #  ####  ####   #####  #### ");
   $display("\nTUD_TESTBENCH:NOTE:SIMPASS: Test PASSed");
end
   $display("Checks run     = %0d", checkcount);
   $display("Errors         = %0d", errorcount);
end
endtask // printTestcaseResults
```

- Messung der Abdeckung des Quellcodes durch die Simulation
  - Block Coverage
    - Abdeckung von Code Blöcken (begin … end)
  - Expression Coverage
    - Abdeckung von Ausdrücken
  - Toggle Coverage
    - Abdeckung von Signalwechseln
  - FSM Coverage
    - Abdeckung von Zuständen (state) und Zustandsübergängen (arc)

- **ACHTUNG: Die Coverage Analyse prüft nicht ob die Ausgabe der Schaltung geprüft wurde.**

Beispiel: FSM Coverage



Beispiel: Block Coverage

Test : test

Instance	Block	Expression	Toggle
tb_verilog_tutorial.i_fsm	95% 20 / 21	92% 11 / 12	92% 12 / 13

File: /home/hoeppner/ICPRO/p_ice/s_verilog/units/verilog_tutorial/source/rtl/verilog/fsm.v

```
     55                end
     56            end
1    57            S_MEAS: begin
1    58                    if (valid_i==1'b1) begin
     59                            state_nxt=S_COMP;
     60                    end
1    61                    else begin
     62                            state_nxt=S_MEAS;
     63                    end
     64            end
1    65            S_COMP: begin
0    66                    if (neg_i==1'b1) begin
     67                            state_nxt=S_MEAS;
     68                    end
1    69                    else begin
     70                            state_nxt=S_DONE;
     71                    end
     72            end
1    73            S_DONE: begin
     74                    state_nxt=S_IDLE;
```

Coverage Report: Uncovered Blocks

Marking:

```
        Instance name: tb_verilog_tutorial.i_fsm
        Module/Entity name: fsm
        File name: /home/hoeppner/ICPRO/p_ice/s_verilog/units/verilog_tutorial/source/rtl/verilog/fsm.v
        Number of uncovered blocks: 1 of 21
        Number of blocks marked COV: 0
        Number of blocks marked IGN: 0

index   uncovered block line no.      line origin description
-------------------------------------------------------------------------
(   9)      66     true part of          66   if (neg_i==1'b1) begin
```

Beispiel: Expression Coverage

Instance	Block	Expression	Toggle
tb_verilog_tutorial.i_fsm	95% 20 / 21	92% 11 / 12	92% 12 / 13

File: /home/hoeppner/ICPRO/p_ice/s_verilog/units/verilog_tutorial/source/rtl/verilog/fsm.v

```
Line
 57                 S_MEAS: begin
 58                         if (valid_i==1'b1) begin
 59                                 state_nxt=S_COMP;
 60                         end
 61                         else begin
 62                                 state_nxt=S_MEAS;
 63                         end
 64                 end
 65                 S_COMP: begin
50% → 66                        if (neg_i==1'b1) begin
 67                                 state_nxt=S_MEAS;
 68                         end
 69                         else begin
 70                                 state_nxt=S_DONE;
 71                         end
```

Coverage Report: Uncovered Expressions Marking:

```
        Instance name: tb_verilog_tutorial.i_fsm
        Module/Entity name: fsm
        File name: /home/hoeppner/ICPRO/p_ice/s_verilog/units/verilog_tutorial/source/rtl/verilog/fsm.v
        Number of uncovered expressions: 1 of 12
        Number of expr items marked COV: 0
        Number of expr items marked IGN: 0

Line     Index  Coverage         Expression description
-------------------------------------------------------------------------------
   66 (   3)  50% (1/2)         if (neg_i==1'b1) begin

            neg_i == 1'b1
           <-1-->   <--2-->

              index  hit | <1>      <2>
           == --------------------------------
              (   1)    0 | lhs  ==  rhs
```

Beispiel: Toggle Coverage (Stack)

Coverage Report: Uncovered Toggles  Marking: 

```
        Instance name: tb_verilog_tutorial.stack_i
        Module/Entity name: stack
        File name: /home/hoeppner/ICPRO/p_ice/s_verilog/units/verilog_tutorial/source/rtl/verilog/stack.v
        Number of signal bits fully toggled: 20 of 37
        Number of signal bits partially toggled(rise): 11 of 37
        Number of signal bits partially toggled(fall): 0 of 37
        Number of signal bits marked COV: 0
        Number of signal bits marked IGN: 0
```

Hit(Full)	Hit(Rise)	Hit(Fall)	Signal
0	0	0	din_i[7]
0	0	0	din_i[6]
0	0	0	dout_o[7]
0	0	0	dout_o[6]
0	1	0	dout_o[5]
0	1	0	dout_o[3]
0	1	0	dout_o[2]
0	1	0	dout_o[1]
0	1	0	dout_o[0]
0	1	0	ptr_r[2]
0	0	0	do_r[7]
0	0	0	do_r[6]
0	1	0	do_r[5]
0	1	0	do_r[3]
0	1	0	do_r[2]
0	1	0	do_r[1]
0	1	0	do_r[0]

Strategien zur Coverage Driven Verification

- Vollabdeckende Simulation
  - Simulation aller Eingangsvektoren bei kombinatorischen Schaltungen: n-Inputs → 2^n Checks
    - z.B. alle Eingangskombinationen der FSM Logik
  - Vorteile:
    - Sehr einfacher Testcase bei vollabdeckender Verifikation
  - Nachteile:
    - Lange Laufzeit bei vielen Inputs
    - Schwierig bei sequentiellen Schaltungen

- Guided Testcases
  - Explizites Stimulieren der Schaltung, insbesondere von Grenzfällen („corner cases")
  - Vorteile:
    - Hohe Coverage möglich
  - Nachteile:
    - setzt detaillierte Kenntnis der Schaltung voraus
    - große Anzahl von Testcases → aufwändiges Setup

- Random Testcases
  - Zufälliges Stimulieren der Schaltung
  - Einschränken des Zufallsbereichs auf zulässige Werte und Abfolgen („constrained random“)
    - z.B. deterministisches Starten (start_i) einer sequentiellen Schaltung bei zufälligen Eingangswerten (din_i)
    - Vorteile:
      - wenige Testcases für hohe Coverage
      - geeignet für kombinatorische und sequentielle Schaltungen
      - Simulationslaufzeit erhöht Coverage
    - Nachteile:
      - Corner Cases sehr unwahrscheinlich

- **→ Praktisch meist Kombination aus mehreren Methoden verwendet.**

- **Komplexe Designs haben viele Testcases**
- Regressions
  - Self-Checking Testcases
  - Automatisierte Simulation
  - Parallelisierung möglich
  - Automatisch erzeugter Report
  - Coverage Analyse
- Vorteile:
  - Automatisierte Verifikation
  - „Messung" des aktuellen Verifikationsstatus → „Management Report"
  - Schnelle Wiederholung der Verifikation bei
    - Entwurfsänderungen
    - Bug-Fixes
    - RTL→ Synthese→ Place&Route

Regression: fast_reg

Regression Group: *default*

Simulation Run *Unit/Simulator/Testcase [Variant]*	State *Results, Pass/ Fail*	Simulator *Defines/ Params*
blizzard_top_tc_avfs_bypass_mode	finished simulation	
blizzard_top/ncsim/tc_avfs_bypass_mode [*default*]	C:2 W:1 N:1 ?	
blizzard_top_tc_avfs_bypass_mode_pmbus	finished simulation	
blizzard_top/ncsim/tc_avfs_bypass_mode_pmbus [*default*]	C:2 W:1 N:1 ?	
blizzard_top_tc_avfs_closed_loop	finished simulation	
blizzard_top/ncsim/tc_avfs_closed_loop [*default*]	C:2 W:1 N:1 ?	
blizzard_top_tc_clkmeas *primary*	finished simulation	
blizzard_top/ncsim/tc_clkmeas [*default*]	C:2 W:1 N:1 ?	
blizzard_top_tc_dhry_programmable	finished simulation	
blizzard_top/ncsim/tc_dhry_programmable [*default*]	C:2 W:1 N:1 ?	
blizzard_top_tc_finish_led	finished simulation	
blizzard_top/ncsim/tc_finish_led [*default*]	C:2 W:1 N:1 ?	
blizzard_top_tc_hpm_bypass_mode	finished simulation	
blizzard_top/ncsim/tc_hpm_bypass_mode [*default*]	C:2 W:1 N:1 ?	
blizzard_top_tc_libtest	finished simulation	
blizzard_top/ncsim/tc_libtest [*default*]	C:2 W:1 N:1 ?	
blizzard_top_tc_pll_change	finished simulation	
blizzard_top/ncsim/tc_pll_change [*default*]	C:2 W:1 N:1 ?	
blizzard_top_tc_power_load_ctrl	finished simulation	
blizzard_top/ncsim/tc_power_load_ctrl [*default*]	C:11 W:10 N:1 ?	
blizzard_top_tc_sanity	finished simulation	
blizzard_top/ncsim/tc_sanity [*default*]	C:2 W:1 N:1 ?	

Zusammenfassung und Ausblick

- Zusammenfassung:
  - Übersicht Design Flow
  - Konzepte zur Hardwarebeschreibung
  - Grundlagen der HDL Verilog
  - Strategien zur Verifikation durch Simulation

- Ausblick auf weitere Themen:
  - Synthesegerechte Verilog Beschreibung
  - RTL-to-GDS Flow (Synthese, Place&Route, Timing Analyse)
  - Erweiterte Verifikationsmethoden

- **→ Vorlesung und Praktikum „VLSI Prozessorentwurf“**